MapReduce Project Report

郭瑛璞 12111408 郭城 12112504

● Project 问题描述

本 project 的数据有两类——逐笔委托数据（股票市场的委托买入和卖出订单信息），以及逐笔成交数据（股票市场实际订单的买入或卖出）。委托订单包含限价单、市价单和本方最优三种不同订单类型。限价单的交易情况为全部成交或全部撤销，市价单的交易则可能出现不同档位的成交价格，以及出现部分成交部分撤销的情况。

对于市价单，通过订单索引在逐笔成交数据中找到其成交的价格种类，并由此倒推市价单的交易情况。最后，根据委托数据和成交数据，按时间顺序，模拟撮和交易所的成交记录。

本 project 的任务是使用 hdfs+mapreduce 的方式确定市价单的价格档位并输出模拟撮合后的结果。

● 整体方案

① 过滤逐笔委托

从逐笔成交数据中选择平安银行连续竞价阶段的数据，并提取有用的字段记为

Order。依据 OrderType 分离出市价订单数据 MarketOrder 和限价订单数据 LimitedOrder，以及本方最优 SpecOrder。Order 中的字段包含委托时间、委托价格、委托数量、买卖方向、委托类型以及委托索引。

② 过滤逐笔成交

从逐笔委托数据中选择平安银行连续竞价阶段的数据，并提取有用的字段记为 Trade。依据 ExecType 分离出成交数据 Traded 和撤单数据 Cancel。

③ 计算市价单成交价格

对于市价单，通过委托索引在成交数据中匹配对应的买/卖方委托索引，找到市价单成交价格的种类，新增字段 Market_Order_Type 记为 K。若成交数据没有匹配到同样的委托索引，则该市价单未成交，即 K=0。

④ 合并数据，排序输出

将计算 K 值后的市价单与限价单和撤单合并，按照时间进行排序，输出最终结果。

● 难点分析

① K 值计算

由于市价单存在许多交易类型，在计算 K 值时需要将委托索引与已成交的订单中进行匹配。

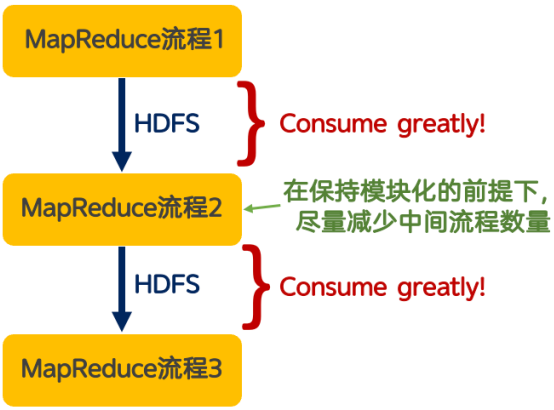
② 输出内容即顺序

对于限价单，存在全部撤销和全部成交的可能，对于限价单，成交价格 K 可能有多种值。本 project 需要将所有的限价委托单、所有标记价格后的市价委托单、所有本方最优订单以及所有的撤单数据合并，按照时间顺序输出结果。

由于 mapreduce 之间有 shuffle 过程，若把时间作为 key 值传入 reducer 中，那么数据已经按照成交时间的升序排好，仅需在同一时间里把按照数据的委托索引进行排序即可。

对于同一时间的委托单和撤单数据，按照委托索引的升序排列。对于同一订单在同一时间出现了委托和撤单的情况，委托单的排序要在撤单之前。

③ 运行效率效率提升



本次 project 数据共 5.05G，我们发现在运行过程中 IO 的时间占了很大部分。在初步方案中我们将原始数据上午下午分别计算，之后进行一层 Map Reduce 的合并，增加的这层本地运行时间在 2 秒以内，但是相比目前方案（上下午使用 MultipleInput）增加的文件 IO 使整个程序在集群上运行时间增加了**一分钟**，可能原因是集群 hdfs 需要进行文件分布式处理，读写效率远低于本地，因此在实现过程中需要及时把不需要的字段删除。通过多输入和多输出的方式减少 mapreduce 总任务的个数，达到加快运行效率的办法。

● 代码实现

Step1 逐笔委托单过滤

Map：

```java
public class Filter_Order_Mapper extends Mapper<LongWritable, Text, Text, Text> {
    private Text filtered = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
       // 请完成这部分代码
        String Line = value.toString();
        String[] split = Line.split( regex: "\t");
        String space = ",";

        if( (!split[8].equals("000001") ) ||
                (Integer.parseInt(split[12].substring(8)) < 93000000) ||
                ((Integer.parseInt(split[12].substring(8)) > 113000000)&&
                    (Integer.parseInt(split[12].substring(8)) < 130000000) )||
                (Integer.parseInt(split[12].substring(8)) > 145700000)){//平安银行
            return;
        }
        else {//筛选出平安银行中9:30-11:30的订单
            //委托索引(7) 委托价格(10) 委托数量(11) 委托时间(12) 买卖方向(13) 委托类型(14)
            filtered.set(split[12] + ","+ split[10] + "," + split[11] + "," + split[13]
                    +"," + split[14] + "," + split[7] + "," + "0" + "," + "2");
            context.write( filtered, new Text(split[14])); //...+Order_type(委托类别作为value)
        }
    }
}
```

应用||条件比&&条件筛选效率更高，将非平安银行的数据以及平安银行数据中非连续竞价的阶段数据剔除，剩余数据为平安银行中上午和下午连续竞价阶段的委托单。选取其中有用的字段（委托时间 委托价格 委托数量 买卖方向 委托类型 委托索引）。

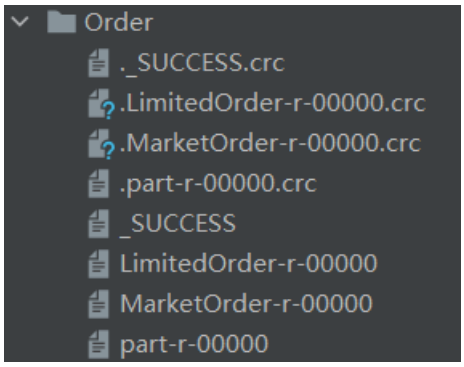
为将格式标准化，先将 MARKET_ORDER_TYPE 全部设置为 0，Cancel_Type 为 2，因此 map 处理后的数据输出后得到的 key 格式为（委托时间，委托价格，委托数量，买卖方向，委托类型，委托索引，0，2），value 为委托类型（1=市价，2=限价，U=本方最优）

Reduce：

每一笔订单都对应唯一的 key 值，根据 value 的委托类型（1=市价，2=限价，U=本方最优）对订单进行分类。

```java
@Override
protected void reduce(Text key, Iterable <Text> values ,
                      Context context) throws IOException, InterruptedException {
    //委托索引 委托价格 委托数量? 委托时间 买卖方向
    //委托时间 委托价格 委托数量 委托方向 委托类型 委托索引 "0" "2"
    for(Text value : values){
        if (value.toString().equals("1")) {//市价单
            multipleOutputs.write( namedOutput: "MarketOrder", key, new Text());
        }
        else if (value.toString().equals("2")){//限价单
            multipleOutputs.write( namedOutput: "LimitedOrder", key, new Text());
        }
        else{// 本方最优"U"
            multipleOutputs.write( namedOutput: "SpecOrder", key, new Text());
        }
    }
}
```

运用多输出格式，把市价单输出到 MarketOrder，限价单输出到 LimitedOrder，本方最优输出到 SpecOrder。



输出结果显示，在连续竞价阶段，平安银行并不存在本方最优订单。在市价单和

```
20190102093000000,9.490000,3000,2,2,133691,0,2,
20190102093000000,9.690000,3000,2,2,133844,0,2,
20190102093000010,9.370000,2300,2,2,134673,0,2,
20190102093000010,9.370000,2400,2,2,134671,0,2,
20190102093000010,9.370000,3200,2,2,134661,0,2,
20190102093000010,9.370000,400,2,2,134663,0,2,
20190102093000010,9.370000,400,2,2,134665,0,2,
20190102093000010,9.370000,400,2,2,134681,0,2,
20190102093000010,9.370000,400,2,2,134689,0,2,
20190102093000010,9.370000,500,2,2,134584,0,2,
20190102093000010,9.370000,500,2,2,134677,0,2,
20190102093000010,9.370000,500,2,2,134679,0,2,
20190102093000010,9.370000,5200,2,2,134586,0,2,
20190102093000010,9.370000,600,2,2,134653,0,2,
20190102093000010,9.370000,800,2,2,134691,0,2,
20190102093000010,9.370000,900,2,2,134638,0,2,
20190102093000010,9.370000,900,2,2,134669,0,2,
20190102093000010,9.390000,200,2,2,134496,0,2,
20190102093000020,9.370000,2100,2,2,134930,0,2,
20190102093000020,9.370000,400,2,2,134893,0,2,
20190102093000020,9.370000,400,2,2,134928,0,2,
```

限价单中，输出格式如图所示

输出格式和标准格式相同，便于后续处理，并且限价单已经按照时间进行了初步排序。而对于市价单，所有的输出 K 暂且都设置成 0，等待后续计算。

Step2 逐笔成交单过滤

Map：

```java
public class Filter_Trade_Mapper extends Mapper<LongWritable, Text, Text, Text> {
    private Text filtered = new Text();
    private final String space = ",";

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
        // 请完成这部分代码
        String Line = value.toString();
        String[] split = Line.split( regex: "\t");

        if((!split[8].equals("000001")) ||//平安银行
                (Integer.parseInt(split[15].substring(8)) < 93000000) ||
                ((Integer.parseInt(split[15].substring(8)) > 113000000)&&
                        (Integer.parseInt(split[15].substring(8)) < 130000000) )||
                (Integer.parseInt(split[15].substring(8)) > 145700000)){//收盘后
            return;
        }
        else {
            //委托时间 委托价格 委托数量 买方索引 卖方索引
            filtered.set(split[15] + space + split[12] + space + split[13]
                    +space + split[10] + space + split[11]);
            context.write(new Text(split[14]), filtered);//成交类别(4或F)作为key
        }
    }
}
```

与委托单的过滤 mapper 类似，应用||条件比&&条件筛选效率更高，将非平安银

行的数据以及平安银行数据中非连续竞价的阶段数据剔除，剩余数据为平安银行中上午和下午连续竞价阶段的成交单。选取其中有用的字段（委托时间 委托价格 委托数量 买方索引 卖方索引）。

将输出 key 值设置为 i 成交类别（F=成交，4=撤销），value 值设置为（委托时间 委托价格 委托数量 买方索引 卖方索引）的格式。

Reduce:

Reducer 接受的 key 共两种，分别为（F=成交，4=撤销），可迭代的 values 则是所有成交订单以及撤销订单的（委托时间 委托价格 委托数量 买方索引 卖方索引）的 list。

对于成交单（Traded）

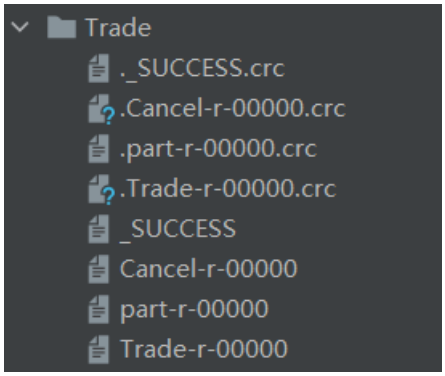
```
if (key.toString().equals("F")) { //成交单(成交时间不需写进去)，需要用委托时间
    for(Text value: values) {
        String[] split = value.toString().split( regex: ",");//委托时间 委托价格 委托数量 买方索引 卖方索引
        Text buy = new Text( string: split[3] + "," + split[1]);//买方委托索引+价格
        Text sale = new Text( string: split[4] + "," + split[1]);//卖方委托索引+价格
          multipleOutputs.write("TradedBuy", buy , new Text());
          multipleOutputs.write("TradedSale", sale, new Text());
        multipleOutputs.write( namedOutput: "Trade", buy , new Text());
        multipleOutputs.write( namedOutput: "Trade", sale, new Text());
    }
}
```

将每一笔成交单的买方索引和卖方索引分别写入 Trade 中，输出格式为（买/卖方索引 价格），即每笔成交都被输出为两个带相同价格的 trade（分别是买+价格和卖+价格）。成交单的输出是为了后续市价单 K 值的计算，为了节省 File IO 的时间，仅需输出索引+价格两个数据即可。

对于撤销单（Canceled）

```
else { //撤销单(成交时间需写进去) K=4
    for(Text value: values) {
        String[] split = value.toString().split( regex: ",");//委托时间 委托价格 委托数量 买方索引 卖方索引
        if(split[3].equals("0")){ //买方委托索引=0，即卖单无人购买
            Text out = new Text( string: split[0] + "," + split[1]  + "," + split [2] + "," + "2"
                    + "," + "2" + "," + split[4] +","+ "0" + "," + "1");//已知撤单中不存在市价单
              multipleOutputs.write("CancelSale", out, new Text());
            multipleOutputs.write( namedOutput: "Cancel", out, new Text());
        }
        else{ //卖方委托索引=0，即买单无人售卖
            Text out = new Text( string: split[0] + "," + split[1]  + "," + split [2] + "," + "1"
                    + "," + "2" + "," + split[3] +","+ "0" + "," + "1");//已知撤单中不存在市价单
              multipleOutputs.write("CancelBuy", out, new Text());
            multipleOutputs.write( namedOutput: "Cancel", out, new Text());
        }
    }
}
```

撤销单存在两种情况，买方索引=0 或卖方索引=0。对于买方索引=0 的订单，提取其卖方索引，能够在委托数据中找到同样的索引，反之亦然。输出格式为（委托时间，委托价格，委托数量，委托方向（1=买，2=卖），委托类型（全部为 2），买/卖方索引，“0”，撤单类别（1） ）。



Step3 市价委托单 K 值计算

计算 K 值需要用到成交单（Traded）和市价单（MarketOrder）中的数据，由于两组数据格式不同，因此需要用到不同的 mapper 类处理逻辑，运用多输入方法。

Map：

对于市价单的 map

```
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    String[] fields = value.toString().split( regex: ",");
    //委托索引 委托价格 委托数量 委托时间 买卖方向 Order_type + " o"
    //20190102093000110,0.000000,100,2,1,141610,0,2,,
      Text text = new Text();
      text.set(fields[5]+","+fields[1]+","+fields[2]+","+fields[0]+","+fields[3]+","+fields[4]);
    context.write(new Text(fields[5]), new Text( string: String.join( delimiter: ",", Arrays.copyOfRange(fields,  from: 0,  to: 4)) + " o"));
}
```

K 值的计算需要匹配两组数据的委托索引，因此把 key 值设为市价单委托索引，value 设置为（委托时间，委托价格，委托数量，委托方向，+”o”）作为标识符

对于成交单的 map

```
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    String[] fields = value.toString().split( regex: ",");
    //买(卖)方委托索引 成交价格 + " t"
    context.write(new Text(fields[0]), new Text( string: fields[1] + " t"));//委托索引作为键
}
```

同样把 key 值设置为委托索引，value 设置为委托价格+”t”，t 作为标识符

Reduce：

从 map 中得到市价委托单和成交单两组数据后，相同 key 值（委托索引）的数据被输入同一个 Reducer。对于有市价委托的数据，我们需要计算 K 值：**其最低成交档位**，一个市价委托单的最低成交档位仅与其不同成交价格的数量有关。建立一个价格表，记录不同价格。

因此，对于每个 key 委托索引，需要对其 value 遍历：

如果是委托数据→记录该数据供输出使用，标识该委托索引存在市价委托

如果是成交数据→判断其价格是否已记录，如果是新价格，记录入价格表

最终，如果该委托索引不存在市价委托，程序不会输出，这笔成交数据为限价成交。如果存在市价委托，计算价格表 size 即为最终所需 K 值，结合市价委托数据输出。

Step4 合并数据，排序输出

由于在之前的步骤中，除委托时间外，数据的输出字段格式已经做过标准化，因此只需要把数据进行合并并排序输出。合并标记后的市价单，所有的委托限价单以及所有的撤单数据。

```
FileInputFormat.addInputPath(job,new Path( pathString: "data/output/Order/LimitedOrder-r-00000"));//限价单(包括撤单)
FileInputFormat.addInputPath(job,new Path( pathString: "data/output/MarketFinal/part-r-00000"));//市价委托单(已标记)
FileInputFormat.addInputPath(job,new Path( pathString: "data/output/Trade/Cancel-r-00000"));//撤单数据
```

Map：

```
public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    String[] values = value.toString().split( regex: ",");
    values[1] = values[1].substring(0,5);//委托价格格式标准化
    //1.已标记的市价委托单(无撤单)
    //2.所有撤单(限价单)
    //3.所有限价单

    // 20190102093613610,10.300000,1000,2,2,820735,0,2,
    // 20190102094711690,0.000000,1800,1,1,1568874,1,2,
    Text TIMESTAMP = new Text(values[0]);

    context.write(TIMESTAMP,new Text(String.join( delimiter: ",", Arrays.copyOfRange(values, from: 1, to: 8))));
    //输出结果
    //<委托时间, 委托价格 委托数量 买卖方向 委托类型 委托索引 K 成交类别>
}
```

由于 map 和 reduce 中会有 shuffle 过程，将委托时间作为 key，其余信息作为 value，传入 reducer 的数据已经按照时间的升序进行了初步的排序。因此，reducer 的目

的是把订单在同一时间内进行排序。

Reduce：

接受同一时间的订单。首先将时间的格式进行标准化。

```
String time = key.toString().substring(8,16);// key = 20190102093000000
//2019-01-02 09:30:00.000000
// time = 09300000

// 定义输入字符串的格式
SimpleDateFormat inputFormat = new SimpleDateFormat( pattern: "HHmmssSS");

// 将输入字符串解析为Date对象
Date date = null;
try {
    date = inputFormat.parse(time);
} catch (ParseException e) {
    e.printStackTrace();
}

// 定义输出字符串的格式
SimpleDateFormat outputFormat = new SimpleDateFormat( pattern: "HH:mm:ss.SS");

// 格式化Date对象为字符串
String outputTime = outputFormat.format(date);
Text timestamp = new Text( string: "2019-01-02 " + outputTime +"0000" );
```

在相同时间进行订单的排序，对于撤单和委托单，按照委托索引的升序排序。而对于相同订单既出现委托又出现了撤销，委托单必须在撤单前排序。

利用 TreeMap 实现

TreeMap 是一种数据结构，通常用于实现关联数组或键值对映射。它根据键的自然顺序或根据提供的比较器对键进行排序。

在 Java 中，TreeMap 是一个实现了 SortedMap 接口的类，它使用红黑树实现，确保了键的有序性。

```
TreeMap<IntWritable,Text> tradeOrder = new TreeMap<>();
TreeMap<IntWritable,Text> cancelOrder = new TreeMap<>();
```

tradeOrder 和 cancelOrder 分别用来存储同一时间的成交单和撤单数据。

```
//考虑同一时间委托并撤单的情况(共6单)
//即在同一时间有两个orderid，分别是委托单(cancelType = 2)和撤单(cancelType = 1)
for (Text value: values){
    String[] info = value.toString().split( regex: ",");
    IntWritable orderId = new IntWritable(Integer.parseInt(info[4]));
    if (Integer.parseInt(info[6]) == 2) { //委托单存在trade
        tradeOrder.put(orderId, new Text(value));
    } else if (Integer.parseInt(info[6]) == 1) { //撤单
        cancelOrder.put(orderId,new Text(value));
    }
}
```

将委托单和撤单的索引作为 key 分别存在 tradeOrder 和 cancelOrder 中，避免了相同索引重复并替代的情况。

```
//委托单先输出
for (Map.Entry<IntWritable, Text> entry : tradeOrder.entrySet()) {
    context.write(timestamp, entry.getValue());
}

//撤单后输出
for (Map.Entry<IntWritable, Text> entry : cancelOrder.entrySet()) {
    context.write(timestamp, entry.getValue());
}
```

为确保数据的输出顺序，将委托单先输出，撤单后输出。排序后的结果为同一之间之内，委托单的索引从小到大+撤单的索引从小到大。这样既保证了委托索引的顺序，又保证了对于同一时间同一订单的委托和撤单，撤单排在委托单之后。